Opinnäytetyö (AMK)

Kone- ja tuotantotekniikka

Koneautomaatio

2018

Jonne Alanne

LAATIKONMUODOSTAJAN PLC-OHJELMAN KONVERSIO SULAUTETULLE JÄRJESTELMÄLLE

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Kone- ja tuotantotekniikka | Koneautomaatio

2018 | 27

Teppo Mattsson

Jonne Alanne

LAATIKONMUODOSTAJAN PLC-OHJELMAN KONVERSIO SULAUTETULLE JÄRJESTELMÄLLE

Click here to enter text.

Opinnäytetyön tavoitteena oli kirjoittaa Jomet Oy:n laatikonmuodostajan ohjelmakoodi sulautetulle järjestelmälle soveltuvaksi ja selvittää, onko mahdollista hyödyntää halvempaa piirikorttiohjausta yleisesti käytetyn ohjelmoitavan logiikan sijaan. Ohjelmakoodin perustana käytettiin valmiin laatikonmuodostajan tikapuulogiikkaa, joka käännettiin Arduinon piirikorttiin sopivaksi strukturoiduksi tekstiksi.

Työssä tutustutaan piirikorttiohjaukseen ja mikrotietokoneisiin yleisesti, sekä ohjataan perusperiaatteet C-kielisen koodin suunnitteluun piirikortille. Työn lopussa on valmis piirikortille soveltuva ohjelmakoodi, jolla laatikonmuodostajan työkierto voidaan toteuttaa. Tarkoituksena on kertoa PLC-ohjelmoinnin ja tekstipohjaisen ohjelmoinnin eroista ja opastaa käyttäjälle peruskomennot sekä toimintaperiaate, jotta käyttäjä voi itse helpommin perehtyä piirikorttiohjaukseen, eikä niinkään käydä läpi itsessään koodin kirjoitusprosessia.

Laatikonmuodostajan ohjelmakoodi pystytään suorittamaan piirikorttiohjauksella, kun tikapuuohjelmoinnin peruskomennot saadaan integroitua sulautetun järjestelmän koodiin. Helppokäyttöisyydessä, muokattavuudessa ja suorituskykyä varmistettaessa piirikorttiohjaus ei ole kuitenkaan järkevä vaihtoehto monimutkaisempien kokonaisuuksien ohjauksessa, ja sulautetut eivät tarjoa tarpeeksi etuja verrattuna nykyaikaisiin PLC-laitteisiin

ASIASANAT:

Sulautettu järjestelmä, Piirikorttiohjaus, Strukturoitu teksti, Ohjelmointi

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Mechanical and Production Engineering | Machine Automation

2018 | 27

Teppo Mattsson

Jonne Alanne

PACKAGING MACHINES PLC-PROGRAM CONVERSION TO AN EMBEDDED SYSTEM

Click here to enter text.

The purpose of thesis was to write a program code compatible for the embedded system of a packaging machine for Jomet Ltd and to examine the possibility to benefit from the more affordable logic board control compared to PLC control. The ladder diagram of the existing packaging machine was used as the basis of the work and thus compiled to a structured text applicable to Arduinos logic board.

The thesis studies logic board control and microcomputers in general as well as guides the basic principles of writing C language based program code for an embedded system. The appendix of the thesis includes a complete program code made for the logic board of the packaging machine which it can use to perform its task cycle. The intention was not to go through the process of writing the code, but to compare the differences between PLC and text based programming and guide the user through the basic principles and commands to help the user to get accustomed to logic board control.

The packaging machine's program can be executed by logic board control when the basic functions of ladder diagram programming are implemented in the embedded system's structured text. When comparing user-friendliness, modifying capability and performance, logic board control is not a reasonable option when controlling more complicated systems. Embedded systems do not offer enough benefits compared to the modern programmable logic controllers.

KEYWORDS:

Embedded system, Logic board control, Structured text, Programming

SISÄLTÖ

LIITTEET

Liite 1. Ohjelmakoodi

KUVAT

KÄYTETYT LYHENTEET TAI SANASTO

Arduino	Mikrokontrolleri-alusta ja ohjelmointiympäristö.
FBD	Function Block Diagram, toimintolohkokaavio.
plcLib	Arduinoon lisätty kirjasto, joka lisää ohjelmaan PLC-ohjelmoinnissa käytettäviä komentoja.
LD	Ladder Diagram, tikapuukaavio.
ST	Structure Text, Strukturoitu teksti.
PC	Personal Computer, henkilökohtainen tietokone.
PLC	Programmable Logic Controller, ohjelmoitava logiikka kontrolleri.
RAM	Random Access Memory, luku-kirjoitusmuisti.
ROM	Read Only Memory, lukumuisti.
Input	Tulopiiri.
Output	Lähtöpiiri.
Sketch	Sketsi, nimitys koodille, jonka Arduino lataa piirilevylle ja josta se suorittaa ohjelman.
Loop	Silmukka jonka sisällä olevaa koodia ohjelma suorittaa toistuvasti.

1 JOHDANTO

Mikrotietokoneet jaetaan toiminnan perusteella joko henkilökohtaiseen tietojenkäsittelyyn tarkoitettuihin tietokoneisiin tai sulautettuihin järjestelmiin, joita käytetään monimuotoisissa tehtävissä, taskulaskimista teolliseen ohjaukseen. Toimintaperiaatteesta huolimatta mikrotietokoneiden pääosat ovat hyvinkin samanlaisia. Ne sisältävät aina mikroprosessorin, muistit ja liitäntäpiirit, sekä komponenttien väliset väylät. Mikroprosessori ohjaa laitteen toimintaa muisteihin kirjoitetun ohjelman ja ulkoisten signaalien mukaisesti. Liitäntäpiireistä kulkeutuu käskyt toimilaitteille tai tiedot prosessorille ulkoisilta antureilta.

Teollisessa ohjauksessa käytetään nykyään paljolti ohjelmoitavaa logiikkaa (PLC, Programmable Logic Controller), jotka ovat korvanneet vanhat releohjaukset, sekä sulautetut järjestelmät. Logiikoilla on omat ohjelmointijärjestelmänsä, jotka käyttävät yleensä tikapuuohjelmointia. Tikapuuohjelmoinnilla tarkoitetaan sitä, että jos ehto toteutuu, laskeudutaan seuraavalle askelmalle, kunnes koko ohjelmarakenne on suoritettu.

Onko nykypäivän automaatioprosessin ohjauksessa kuitenkaan aina järkevintä käyttää PLC-laitteita, jotka ovat kalliimpia verrattuna esimerkin piirikortti-ohjaukseen? Saadaanko sulautetulla järjestelmällä hallittua laatikonmuodostajaa yhtä luotettavasti ja helposti kuin ohjelmoitavalla logiikalla? Entä pystyykö piirikortti suorittamaan luotettavasti ja tehokkaasti siltä ohjelmassa vaaditut tehtävät?

Työ on toteutettu Jomet Oy:lle, joka on vuonna 1977 perustettu automaatioon erikoistunut yritys. Se suunnittelee, rakentaa, integroi ja ylläpitää pakkaukseen, lavaukseen ja materiaalinhallintaan kehitettyjä ratkaisuja. Jomet Oy valmistaa asiakkaalle räätälöityjä automaatioratkaisuja eri teollisuusaloille elintarviketeollisuudesta elektroniikkaan ja päivittäistavaraan.

2 TYÖN TAVOITE

Työn tavoitteena on laatikonmuodostajan ohjelmakoodin kääntäminen strukturoiduksi tekstiksi, jota pystytään käyttämään yksinkertaisemmassa piirikortissa. Ohjelman tarvittavien lähtöarvojen asettaminen, muuttujien määrittäminen sekä lopullisen koodin kirjoittaminen, joka toimii vastaavasti kuten ohjelmoitavalla logiikalla suoritettuna.

Työssä selvitetään, pystytäänkö piirikorttiohjauksella luotettavasti ja riittävän helposti suorittamaan pakkauskoneen ohjelma ja tarjoamaan sen vaatimat edellytykset. Entä riittääkö piirikortin suoritusteho pyörittämään ohjelmaa erilaisesta suoritustyylistä riippumatta?

Piirikortin ohjelmaan ei kuitenkaan opinnäytetyössä kirjoiteta muutettavia parametreja, turvapiirejä, saati koko käyttöliittymää, jonka pakkauskone vaatisi. Varsinkin graafisen käyttöliittymän ohjelmointi on yksi ratkaisevista kysymyksistä piirikorttiohjauksen kannattavuutta tutkiessa, sillä sen merkitys on hyvin suuri käytettävyyttä ajatellen sekä parametrien muuttamista varten. Ilman käyttöliittymää laatikonmuodostaja vaatisi aina ulkoisen ohjelmointilaitteen, kuten tietokoneen, muutoksia tehtäessä. Avoimien lähdekoodien käyttöliittymiä, esimerkiksi 3D-tulostimille, löytyy kyllä ladattavaksi verkosta, mutta niiden muokkaaminen halutunlaiseksi vaatisi suuren työn. Myös piirikortin suoritusteho voi tulla vastaan, sillä C-kieli-pohjaisessa ohjelmoinnissa koodi tekee silmukkaa (loop) loputtoman määrän ja graafisen käyttöliittymän ylläpito varsinaisen ohjelmakoodin suorittamisen rinnalla voi hidastaa laitteen toimintaa merkittävästi tai jopa estää ohjelman luotettavan ajon.

Opinnäytetyön alussa tutustutaan piirikortin perusperiaatteisiin, jonka jälkeen tutustutaan ja ohjeistetaan ohjelmakoodin tekoon, joka on liitteenä työn lopussa.

3 MIKROTIETOKONE

Mikrotietokoneet jaetaan yleiskäyttöisiin eli henkilökohtaiseen tietojenkäsittelyyn tarkoitettuihin tietokoneisiin (*Personal Computer, PC*) ja sulautettuihin järjestelmiin (*Embedded Control, Embedded System*), eli joihinkin erityistoimintaan tarkoitettuihin mikrotietokonelaitteisiin.

3.1 Yleiskäyttöinen mikrotietokone

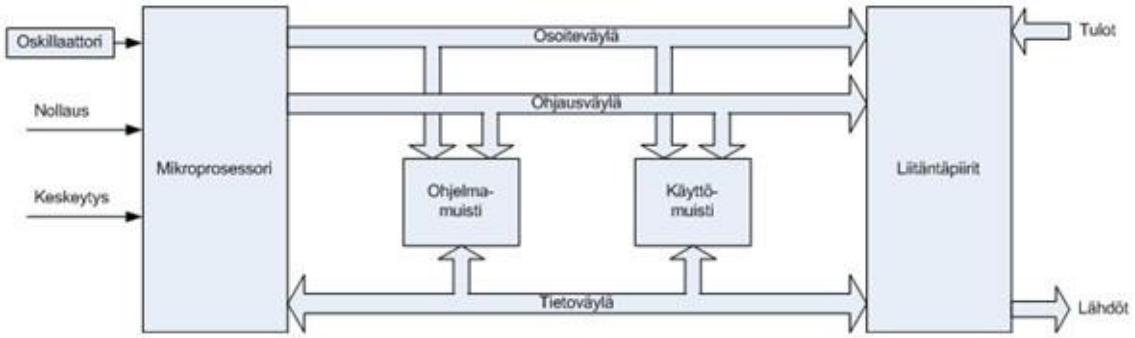
Yleiskäyttöisille mikrotietokoneille on ominaista, että niissä käytetään useita erilaisia ohjelmia. Laitteiden keskusmuisti (*Central Memory*) on yleensä RAM-muistia (*Random Access Memory*), johon tietojen lukemisen lisäksi voidaan tallentaa tietoa. RAM-muistiin tallennuttu tieto kuitenkin häviää käyttöjännitteen katketessa, joten yleiskäyttöiset mikrotietokoneet tarvitsevat apumuistilaitteita, jotta ohjelmat ja käsiteltävä tieto saadaan tallennettua. Tavallisimpia apumuistilaitteita ovat kiintolevyt, levykeasemat ja CD-ROM-asemat. (Koskinen 2004, 7.)

3.2 Sulautettu järjestelmä

Sulautetulla järjestelmällä tarkoitetaan laitetta, jossa mikrotietokone on osana jotakin elektroniikkajärjestelmää ja sitä käytetään johonkin erityistarkoitukseen. Sulautetun järjestelmän muisti jakautuu yleensä fyysisesti kahteen osaan, ohjelmamuistiin (*Program Memory*) ja käyttömuistiin (*Data Memory*). Sulautettujen järjestelmien ohjelma on tallennettu kiinteään ROM-muistiin (*Read Only Memory*), jolle on ominaista, että se säilyttää tiedon käyttöjännitteen katkeamisen jälkeen. Sulautetut järjestelmät eivät yleensä sisällä apumuistilaitteita. (Koskinen 2004, 7.) Ohjelmamuisti on haihtumatonta muistia, joka ei häviä käyttöjännitteen katketessa. Vastaavasti käyttömuisti on haihtuvaa muistia, jonka sisältö häviää jännitteen katketessa. (Koskinen 2004, 8.)

4 RAKENNE JA TOIMINTA

Mikrotietokoneen perusosat ovat mikroprosessori, muistit ja liitäntäpiirit. Näitä osia yhdistää toisiinsa kolme väylää: tietoväylä (*Data Bus*), osoiteväylä (*Address Bus*) ja ohjausväylä (*Control Bus*) (kuva 1).



Kuva 1. Mikrotietokoneen rakenne

Näitä väyliä kutsutaan mikrotietokoneen sisäisiksi signaaleiksi. Sisäisten signaalien avulla mikroprosessori ohjaa kaikkia mikrotietokoneen väyliin liitettävien osien toimintaa. Mikroprosessorille tulee myös signaaleja, jotka ohjaavat prosessorin toimintaa ja vaikuttavat koko mikrotietokoneen toimintaan. Näitä signaaleja kutsutaan ulkoisiksi ohjauksiksi. (Koskinen 2004, 13.)

4.1 Mikroprosessori

Mikroprosessori on mikrotietokoneen tärkein osa. Yksinkertaistetusti sen tehtävänä on lukea muistiin tallennettua ohjelmaa ja suorittaa ohjelmassa määrätyt toimenpiteet. Mikroprosessori lukee tietoa muistipiireistä ja kirjoittaa tietoa muisti- ja liitäntäpiireihin, jotka ohjelmassa on käsketty suorittaa. (Koskinen 2004, 13 –14.)

4.2 Väylät

Mikrotietokoneen sisällä on yleensä kolme väylää. Tietoväylä on sisäinen signaali, jossa kulkee käsiteltävä tieto, osoiteväylällä osoite, mistä tieto haetaan tai mihin se tallennetaan, ja ohjausväylä, joka antaa tietoa siitä, mitä prosessori on tekemässä. Mikrotietokoneen toiminnan kannalta ohjausväylän tieto on tärkeintä informaatiota.

Osoiteväylä on prosessorilta lähtevä yksisuuntainen väylä, jonka avulla prosessori määrää kohdekomponentin ja sieltä muistipaikan, jonka kanssa se asioi. Osoiteväylä kytkekään niihin piireihin, joista prosessori lukee tai kirjoittaa tietoa.

Tietoväylän tehtävä on siirtää ohjelmakoodia ohjelmamuistista prosessorille. Se myös välittää kaiken käsiteltävän tiedon prosessorin, muistien ja liitäntäpiirien välillä. Tietoväylä on kaksisuuntainen ja leveys tavallisesti 4, 8, 16, 32 tai 64 bittiä. Leveys määrää, kuinka suurina paloina prosessori pystyy siirtämään ja käsittelemään tietoa yhdellä muistinosoituksella. Leveydellä on erittäin suuri merkitys mikrotietokoneen tehokkuudelle. Leveys ei ole kuitenkaan este käsiteltävän tiedon koolle, mutta esimerkiksi 8-bittinen prosessori joutuu käsittelemään 32-bittisiä lukuja neljässä osassa. Tietoväylä kytketään kaikille niille piireille, joihin prosessori kirjoittaa tietoa, sekä niille, joista se lukee tietoa. (Koskinen 2004, 14.)

Ohjausväylä antaa tiedon siitä, mitä mikroprosessori on tekemässä. Tämä on huollon ja toiminnan kannalta kaikkein tärkein väylä. Ohjausväylän liitännät ovat yleensä yksisuuntaisia, joista suurin osa on lähtöjä. (Koskinen 2004, 15.)

5 MUISTI

5.1 Muistin jaottelu

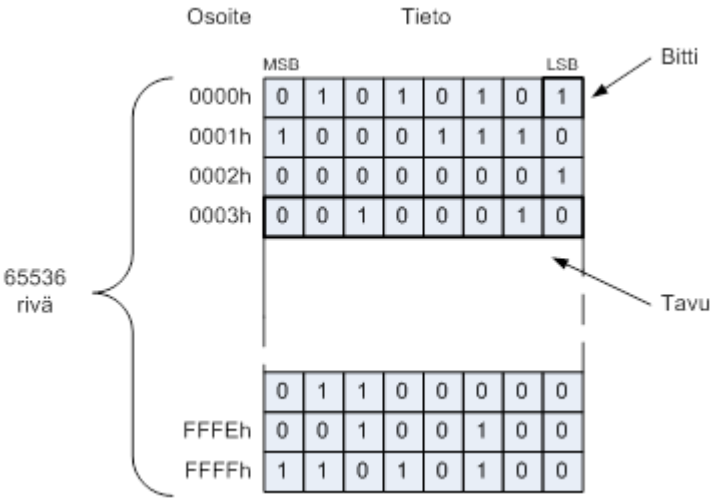
Muistit voidaan jakaa käytön kannalta kahteen ryhmään, ohjelmamuistiin ja käyttömuistiin.

Ohjelmamuistiin (*Program Memory*) tallennetaan ohjelma eli mikrotietokoneen toiminta-ohje. Ohjelmamuisti on yleensä lukumuistia (*Read Only Memory, ROM*). ROM-muistin sisältö pysyy muuttumattomana sen jälkeen, kun se on ohjelmoitu. Ohjelmointi tehdään ohjelmointilaitteella, ja nykyisin lukumuisteja voidaan ohjelmoida uudelleen irrottamatta piiriä laitteesta. Tämä tekee päivittämisestä helpompaa. Kaikki ROM-muistit ovat haihtumattomia, toisin sanoen ohjelma pysyy muistissa käyttöjännitteen katketessa.

Käyttömuisti on luku-kirjoitusmuistia (*Random Access Memory, RAM*). Käyttömuistiin voidaan tallentaa ohjelman tarvitsemien muuttujien arvoja ja tiedon käsittelyssä syntyneitä välituloksia. RAM-muistia voidaan lukea ja kirjoittaa, ja niitä on kahta tyyppiä. Dynaamista muistia (*Dynamic RAM, DRAM*), jota pitää virkistää jatkuvasti, jotta se säilyttää muistissa olevan tiedon, sekä staattista *(Static RAM, SRAM*), jota ei tarvitse virkistää. Molemmat ovat haihtuvia muistityyppejä, toisin sanoen tieto häviää muistista käyttöjännitteen katketessa. (Koskinen 2004, 18.)

5.2 Muistipiirin rakenne ja toiminta

Muistipiiriä voidaan kuvata lokerikkona, jossa jokaisella lokerolla on oma osoite ja jokaiseen lokeroon voi tallentaa tietyn määrän tietoa. Lokeron koko voi olla 8, 16, 32, jne. bittiä, ja jokaiseen muistikennoon voi tallentaa yhden bitin tilan. Bitillä voi olla vain kaksi tilaa, 0 tai 1, eli yksi binääriluku (Kuva 2).



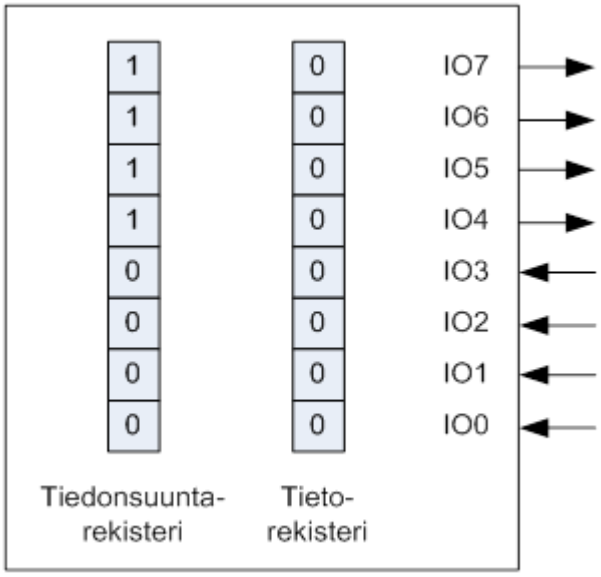
Kuva 2. Muistipiirin rakenne

Vaikka mikrotietokone pystyy käsittelemään tietoa yhden bitin tarkkuudella, yleensä se tallennetaan muistiin vähintään kahdeksan bitin, eli yhden tavun paloina. Tavujen binääriluvun merkitys riippuu asiayhteydestä ja ne voivat olla esimerkiksi ASCII-koodattuja merkkejä tai ohjelmakoodia, jotka prosessori ymmärtää erilaisiksi käskyiksi. Jokaisella prosessorilla on oma konekielensä, jossa jokaista käskyä vastaa oma binääriluku. (Koskinen 2004, 19 –20.)

6 LIITÄNTÄPIIRIT

Liitäntäpiirien avulla mikrotietokone on yhteydessä sen muuhun elektroniikkaan. Liitäntäpiirit jaetaan tulopiiriin (*input)* ja lähtöpiiriin (*output*). Näiden englanninkielisestä nimestä tulee usein käytetty nimitys I/O-piiri. Liitäntäpiirien liitännät ovat yleensä kaksisuuntaisia, ja ne voidaan ohjelmallisesti valita tuloiksi tai lähdöiksi. (Koskinen 2004, 25.)

Yksinkertainen liitäntäpiiri, jonka lähdöt voidaan määritellä tuloiksi tai lähdöiksi, voi sisältää kaksi rekisteriä. Toinen on tiedonsuuntarekisteri *(Data Direction Register*) ja toinen varsinainen tietorekisteri (*Data Register*) (Kuva 3). Tiedonsuuntarekisterissä on yksi bitti jokaista I/O-liitäntää varten, joka määrittää, onko liitäntä tulo vai lähtö. Tietorekisteriä luettaessa vain Outputeilla on merkitystä, ja kirjoittaessa vain Inputteihin kirjoitetaan. (Koskinen 2004, 25.)



Kuva 3. Liitäntäpiirit

7 OHJELMOINTIKIELEN RAKENNE

Työssä on tarkoituksena muuttaa Omronin logiikalle tehty ohjelma Arduinon piirilevylle, joka käyttää C-kielipohjaista ohjelmaa. Suurimpana eroavaisuutena on ohjelmointikielien erilainen rakenne. Omronin ohjelmointijärjestelmä CX-programmer käyttää tikapuurakennetta (Ladder Diagram, LD), joka on hyvin visuaalinen ohjelmointirakenne. Peruspe-riaatteena jokainen ehto täytyy toteutua, ennen kuin ohjelma siirtyy seuraavalle aske-leelle.

Arduinon ohjelmointi perustuu C-kieleen, tekstipohjaiseen sketsiin (sketch). Ohjelman suoritusperiaate on täysin erilainen, ja ohjelma pyörii sketsin ympäri niin kauan kunnes jokin ehto toteutuu ja suorittaa sitä, kunnes palaa takaisin jatkamaan silmukkaa (loop), kun ehto ei enää toteudu.

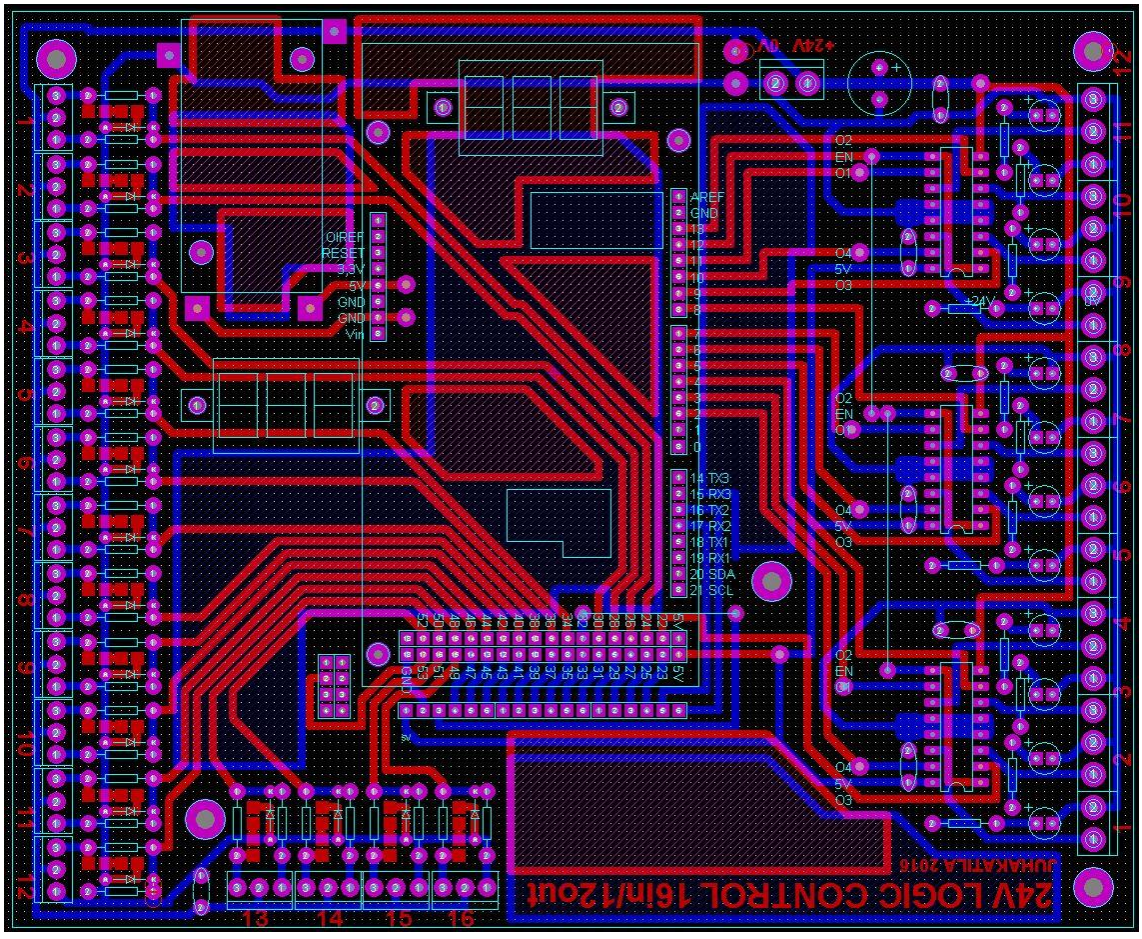
Arduinoon saa kuitenkin lisättyä erilaisia kirjastoja, jotka lisäävät toiminnallisuuksia ohjelmien suorittamiseen, kuten erilaisten moottorien käyttö, Ethernetiin yhdistäminen ja GSM-verkot. Opinnäytetyössä käytetty kirjasto *plcLib* sisältää komentoja, joilla koodista saa muodostettua tikapuurakennetta muistuttavan lisäämällä steppejä koodiin, input/out-put-komentoja, timer-laskureita ja erilaisia ohjelmoitavassa logiikassa käytettäviä ehtoja.

Avoimen lähdekoodin ohjelmisto plcLib tarjoaa hyvät työkalut muuttaessa graafista PLC-pohjaista ohjelmaa strukturoiduksi tekstiksi, sekä sisältää monia työkaluja kuten Func-tion Blockit, jonot (Sequential Fuction Chart) sekä laskurit. Kirjasto ei kuitenkaan itses-sään sisällä graafista suunnittelua tai ohjelman simulointia. (Ditch 2017, 5.)

8 PIIRIKORTIN RAKENNE

Piirilevyn pohjana on Arduino Mega 2560 -mikrokontrolleri, jossa on ATmega 2560 -prosessori, 256 kilotavun ohjelmamuisti (ROM), 8 kilotavun käyttömuisti (RAM), 54 kappaletta digitaalisia tulo/lähtö portteja, 16 analogista tuloporttia, USB-liitin sekä ulkoinen virtalähde. (Arduino Mega2560.)

Piirilevyn on suunnitellut ja valmistanut Juha Katila, ja levyyn on integroitu Arduinon mikrokontrolleri, RepRapDiscount Graphic Smart Controller graafinen kontrolleri 128x64-pikselisellä LCD-näytöllä sekä käännettävällä valintapainike. Piirilevy on varustettu kuudellatoista tulopinnillä, sekä kahdellatoista lähtöpinnillä, joihin on liitetty LED-indikaattori helpottamaan ohjelman suorituksen havainnointia (Kuva 4). Tämä on huomattava helpotus koodia testattaessa, koska lähtöön ei tarvitse asentaa toimilaitteita ja ohjelman ja outputtien testaus onnistuu jopa parilla siirrettävällä rajakytkimellä.



Kuva 4. Piirikortin rakenne

9 VAKIOIDEN ASETTAMINEN

Aina ei ole mahdollista toimia mikrokontrollerin vakiopinnien mukaisesti vaan joudutaan itse määrittämään I/O-konfiguraatio. Näin joudutaan usein tekemään, kun käytössä on useita eri laitteita, I/O-piirien määrä vaihtelee tai halutaan itse vaikuttaa tulojen ja lähtöjen määrään. Ohjelmaan täytyy siis määrittää 16 tulopinniä, 12 lähtöpinniä, kääntöenkooderin liikkeet, stop-painike, sekä piezo-summeri.

Ohjelman alkuun täytyy lisätä lause #define noPinDefs, joka poistaa käytöstä mikrokontrollerin vakiolähdöt. Seuraavaksi määritetään vakiot piirivelykuvan mukaisesti. Esimerkkinä tulopiiri 1, joka piirilevykaavion mukaan liittyy mikrokontrollerin I/O-piirissä pinniin 24. Kirjoitetaan siis vakio (constant), joka nimetään ANTURI1 ja liitetään pinniin 24 lauseella const int ANTURI1 = 24;

Suorituskoodin alkuun täytyy lisätä komento void customIO(); jonka sisään määritetään jokaisen releen ja anturin suunta, eli onko kyseessä lähtö (output) vai tulo (input). Anturi 1 tarvitsee siis lauseen pinMode(ANTURI1, INPUT); jolloin ohjelma tunnistaa anturin tuloksi.

```
#define noPinDefs
#include <plcLib.h>

// tulopinnit:
const int ANTURI1 = 24;
const int ANTURI2 = 26;
const int ANTURI3 = 28;
const int ANTURI4 = 30;
const int ANTURI5 = 34;
const int ANTURI6 = 36;
const int ANTURI7 = 38;
const int ANTURI8 = 40;
const int ANTURI9 = 42;
const int ANTURI10 = 44;
const int ANTURI11 = 46;
const int ANTURI12 = 48;
const int ANTURI13 = 50;
const int ANTURI14 = 52;
const int ANTURI15 = 51;
const int ANTURI16 = 49;
const int STOPPI = 41;
const int ENCODER_VASEN = 31;      //encoder CW
const int ENCODER_OIKEA = 33;      //encoderi CCW
const int ENCODER_PAINO = 35;      //encoderin nappi
```

```
// tulopinnit inputeiksi:
pinMode(ANTURI1, INPUT);
pinMode(ANTURI2, INPUT);
pinMode(ANTURI3, INPUT);
pinMode(ANTURI4, INPUT);
pinMode(ANTURI5, INPUT);
pinMode(ANTURI6, INPUT);
pinMode(ANTURI7, INPUT);
pinMode(ANTURI8, INPUT);
pinMode(ANTURI9, INPUT);
pinMode(ANTURI10, INPUT);
pinMode(ANTURI11, INPUT);
pinMode(ANTURI12, INPUT);
pinMode(ANTURI13, INPUT);
pinMode(ANTURI14, INPUT);
pinMode(ANTURI15, INPUT);
pinMode(ANTURI16, INPUT);
pinMode(ENCODER_VASEN, INPUT);
pinMode(ENCODER_OIKEA, INPUT);
pinMode(ENCODER_PAINO, INPUT);
pinMode(STOPPI, INPUT);
```

Kuva 5. Tulovakioiden määrittäminen

10 PERUSKOMENNOT

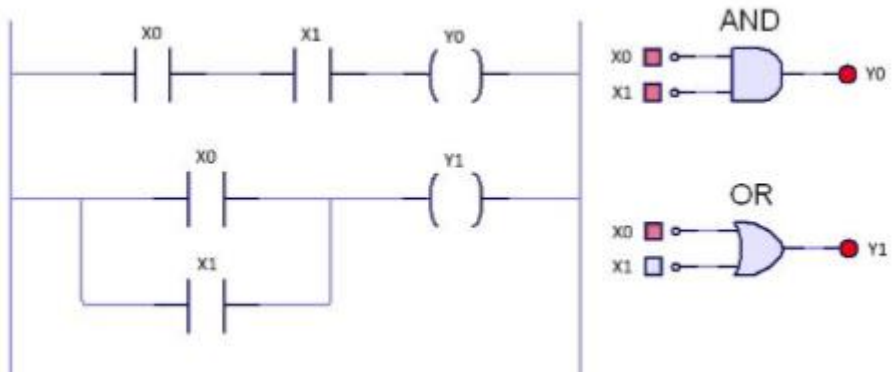
Ohjelman plcLib-kirjastoon on lisätty peruskomentoja, jotka toimivat samaan tapaan kuin graafisissa ohjelmistoissa toimivat Function Blockit.

10.1 Komennot: in ja out sekä andBit, andNotBit ja orBit

Komennot in (input) ja out (output) toimivat kuten useimmissa muissakin ohjelmissa, eli input lukee tulo-komennon, joka voi olla joko koodin asettama ehto tai konkreettinen anturi, kuten rajakytkin. Output määrittää lähtöarvon, joka toimii samalla periaatteella. Lähtöarvo voi olla koodin asettama ehto tai steppi, tai lähtösignaali esimerkiksi moottorille.

Komento andBit vaatii kahden ehdon täyttymisen, kuten alla olevassa kuvassa, kun X0 ja X1 toteutuu, lähtö Y0 vaikuttuu. Komento orBit, eli ’TAI’ funktio vaatii joko X0 tai X1 ehdon toteutumisen, jolloin Y1 lähtö vaikuttuu.

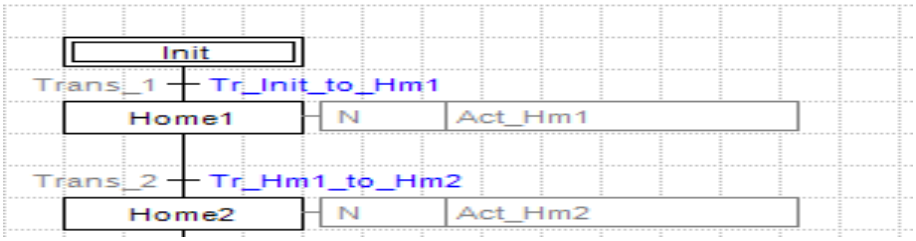
AND	OR
in(X0)	in(X0)
andBit(X1)	orBit(X1)
out(Y0)	out(Y1)



Kuva 6. AND ja OR

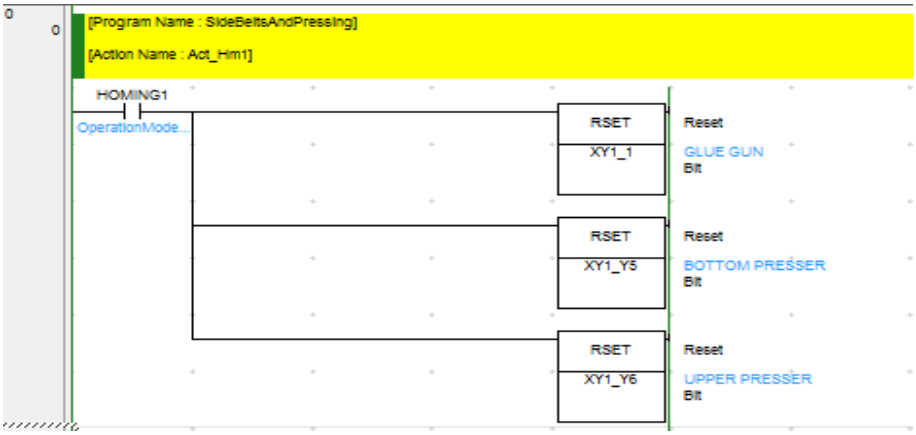
10.2 Komennot: set ja reset

Koska ohjelman suoritus Arduinossa eroaa suuresti tikapuulogiikasta, tarvitsee jokainen ohjelmalohko ajatella omana askelmana (step), ja kun askelman ehdot on toteutettu, siirryttävä seuraavaan askelmaan. Kuten Omronin logiikassa varsinaisia toimia (Act) ja siirtymisehtoja (Transmissions) ei ole, vaan jokainen askeleen pitää sisältää nämä ehdot ja seuraavaan askelmaan siirtyessä nollata (reset) jälkimmäinen askelma, jottei se jää vaikuttamaan ohjelmaan.



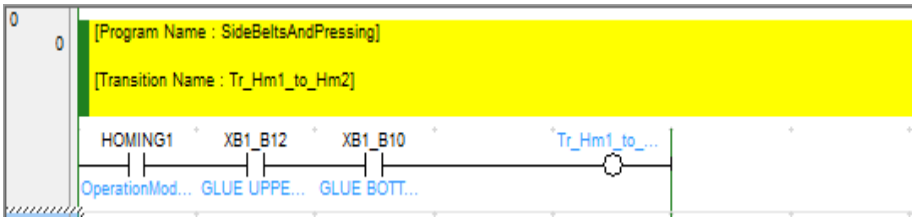
Kuva 7. Tikapuurakenne

Kun tarkastellaan SideBeltsAndPressing ohjelmaa ensin Omronin CX-One tikapuulogiikan perusteella, nähdään, että Home1 blockissa, OperationModeAuto bitin ollessa vaikuttunut, lähdöt XY1_1, XY1_Y5 sekä XY1_Y6 resetoidaan.



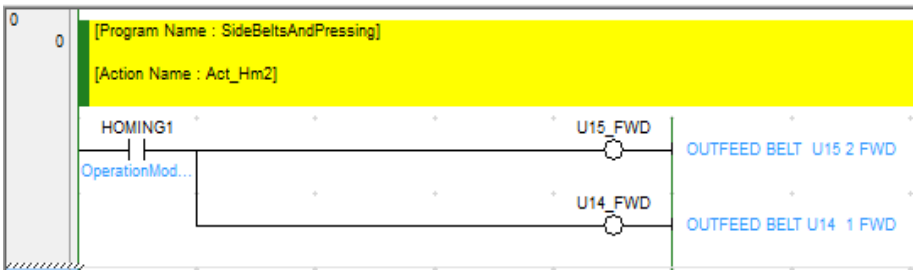
Kuva 8. Action Home1

Transmission Home1-askelmasta Home2-askelmaan toteutuu, jos ehdossa olevat OperationMode_Homing ja tulot XB1_B12 sekä XB1_B10 ovat kaikki vaikuttuneet samanaikaisesti.



Kuva 9. Transmission Home1 to Home2

Home2-lauseessa, jos OperationMode_Homing on vaikuttuneena, lähdöt U14 ja U15 ovat aktiivisia, eli kuljettimet pyörivät eteenpäin.



Kuva 10. Action Home2

Tekstipohjaiseksi ohjelmakoodiksi nämä kolme askelmaa kääntyvät kuvan osoittamalla tavalla.

```
in(SBAP_HOME1);
andBit(OperationMode_Homing);
reset(RELE1); //GLUE GUN (XY1_1)
reset(RELE6); //BOTTOM PRESSER (XY1_Y5)
reset(RELE7); //UPPER PRESSER (XY1_Y6)

in(SBAP_HOME1);
andBit(OperationMode_Homing);
andBit(ANTURI12); //GLUE UPPER PRESSER DOWN
andBit(ANTURI10); //GLUE BOTTOM PRESSER DOWN
set(SBAP_HOME2);
reset(SBAP_HOME1);

in(SBAP_HOME2);
andBit(OperationMode_Homing);
out(RELE12); //OUTFEED BELT U15 2 FWD
out(RELE11); //OUTFEED BELT U14 1 FWD
```

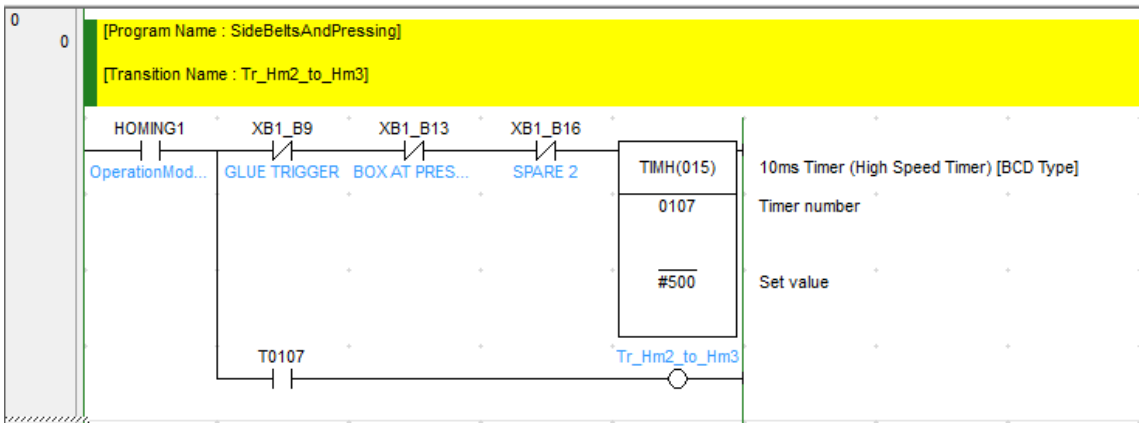
Kuva 11. Sidebelts and Pressing strukturoitu teksti

Arduinon ohjelma lähtee aina suorittamaan koodia ylhäältä alaspäin ja suorittaa looppia loputtomiin. Ensimmäinen kohta (SBAP_HOME1) käyttäytyy täysin samalla tavalla kuin Omronin logiikassakin, mutta toisessa komennossa tulee esille set- ja reset-käskyjen tärkeys. Kun käskyn ehdot ovat täyttyneet, set(SBAP_HOME2); siirtää ohjelman seuraavaan steppiin ja samalla reset(SBAP_HOME1); eli resetoi edeltävän HOME1-askelman. Näillä set ja reset komennoilla tekstipohjaisesta koodista saa tikapuurakennetta muistuttavan ohjelmarakenteen, joka ei siirry suorittamaan seuraavaa komentoa, ennen kuin ehto on toteutunut. Reset-komento siirtymisen jälkeen nollaa edeltävän komennon, jottei se jää taustalle vaikuttuneeksi.

10.3 Komento: timerOn

Monesti ohjaukseen halutaan viiveitä, joko ohjelman suorittamiseen tai häiriöiden ehkäisyyn.

SideBeltsAndPressing ohjelmassa, kun OperationMode_Homing on vaikuttuneena ja inputit XB1_B9, XB1_B13 ja XB1_B16 eivät, ajastin nro. 0107 menee päälle 5000 millisekunniksi, eli viideksi sekunniksi. Tämän jälkeen T0107 ehto vaikuttuu ja ohjelma siirtyy kohtaan HOME3.



Kuva 12. Timer

Alhaalla olevassa kuvassa on kuvattu timerOn-komennon toiminta strukturoidussa tekstissä. Output SPARE2 on jätetty pois selkeytyksen vuoksi, koska sillä ei ole ohjelman toiminnallisuuden kannalta merkitystä. timerOn komento käynnistää laskurin 0107, joka

laskee 5000 millisekunttiin asti, jonka jälkeen output T0107 vaikuttuu ja ohjelma pystyy siirtymään seuraavaan steppiin.

```
in(SBAP_HOME2);
andBit(OperationMode_Homing);
andNotBit(ANTURI9); //GLUE TRIGGER (XB1_B9)
andNotBit(ANTURI13); //BOX AT PRESSING POS (XB1_B13)
timerOn(TIMER0107, 5000);
out(T0107);

in(SBAP_HOME2);
andBit(OperationMode_Homing);
andBit(T0107);
set(SBAP_HOME3);
reset(SBAP_HOME2);
```

Kuva 13. timerOn-komento

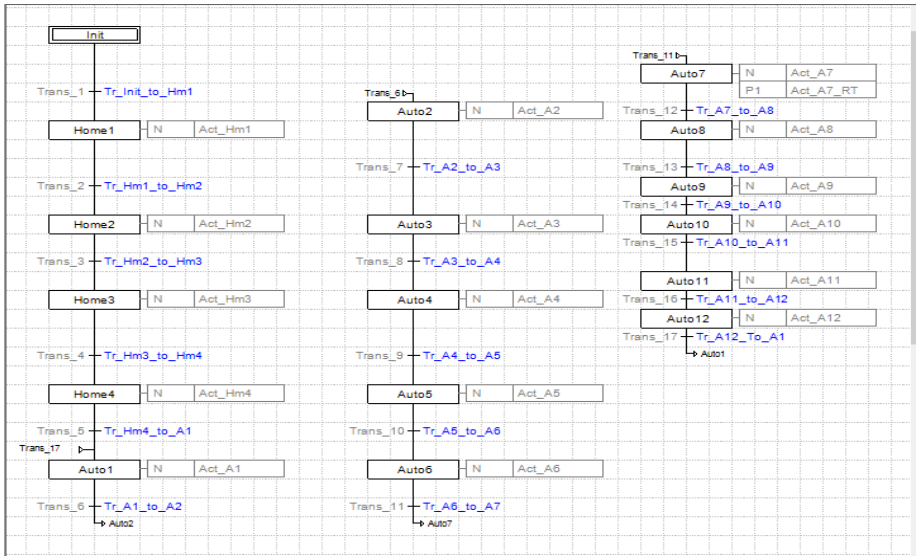
11 OHJELMAKOODI

Pakkauskoneen ohjelma koostuu kolmesta pääosasta; makasiini (Magasin), laatikon muodostus ja kuljetus (Box Open and Move) sekä sivukuljettimet ja puristus (Sidebelts and Pressing). Arduinon koodissa nämä kolme kokonaisuutta suorittavat jokainen omaa silmukkaansa ja nämä kolme *looppia* on sijoitettu pääohjelman *looppiin*.

```
void loop() {

  loopSideBeltsAndPressing();

  loopBoxOpenAndMove();

  loopMagasin();}
```

11.1 Box Open and Move -rakenne

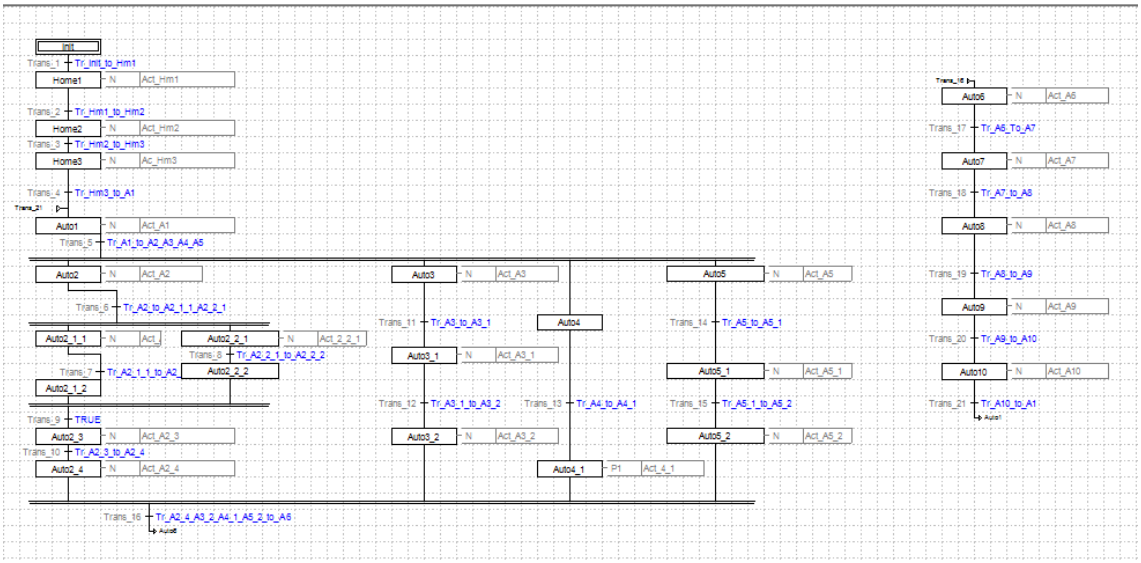
Laatikon muodostus ja kuljetus ovat toiminnaltaan melko suoraviivaisia. Ohjelman alussa on kotiinajo (Home), jossa varmistetaan, että toimilaitteet ovat tarvittavissa positioissa, sekä pakkauskone on valmiudessa suorittaa automaattiajoa (Auto). Laatikonmuodostuksessa varmistetaan avaamattomien laatikoiden saatavuus makasiinista, imukupeilla ja kynsillä laatikoiden avaus, sekä kuljetus liimaukseen. Laskureilla saadaan toiminnoille varmuutta ja tarvittavat suoritusajat.



Kuva 14. Box Open and Move -rakenne

11.2 Sidebelts and Pressing -rakenne

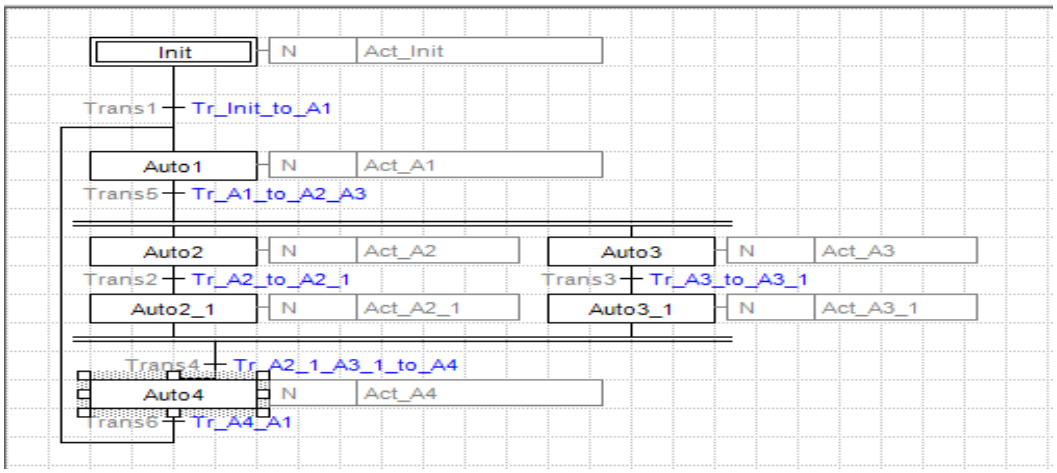
Sivukuljetinten ja puristuksen ohjelmassa haasteen aiheuttavat haarautuvat ohjelmat, kuten koneen valmius vastaanottaa laatikko, molempien puolien kuljettimet sekä oikean sijainnin saavuttaminen liimapistooleille. Kun edellä mainitut vaiheet ovat valmiita, suoritetaan laatikon puristus ja valmiin laatikon siirto pois pakkauskoneesta.



Kuva 15. Sidebelts and Pressing -rakenne

11.3 Magasin -rakenne

Makasiinilla tarkoitetaan syöttöpöytää, johon käyttäjä asettaa avaamattomia laatikon aihioita. Ohjelmallisesti hyvin yksinkertainen koodi varmistaa, että aihioita on saatavilla koneeseen, ja pyörittää kuljettimia eteenpäin, jos laatikkoa ei tunnisteta. Vasemmalle ja oikealle puolelle on omat tunnistimensa ja vaiheensa, jotta aihio ei jää poikittain kuljettimelle.



Kuva 16. Magasin -rakenne

12 YHTEENVETO

Laatikonmuodostajan ohjelmakoodi pystytään suorittamaan piirikorttiohjauksella, kun tarvittavat tikapuuohjelmoinnin peruskomennot saadaan integroitua koodiin. C-kieli-pohjaisen ohjelmoinnin ja periaatteiden opettelu vaatii tietenkin perehtymistä, mutta strukturoidulla tekstillä saadaan suoritettua ohjelma kuitenkin halutulla tavalla. PLC-laitteissa graafisuus on kuitenkin hyvin suuressa roolissa ja tätä ei pystytä käyttämään tekstipohjaisessa ohjelmoinnissa, joten käyttäjälle on haastavampaa ymmärtää koodin toiminta. Ohjelmoitavat logiikat tarjoavat myös simulointimahdollisuuden ja online-vianhaun sekä -muokkauksen, jotka ovat varsinkin häiriötilanteissa sekä virheenratkaisussa oleellisia toimivuuden kannalta, piirikorttiohjauksessa tällaista mahdollisuutta ei ole.

Käyttöliittymän rakentaminen piirikortille on myös hyvin haastavaa ja kriittinen kysymys kannattavuutta ajateltaessa. Parametreja ja arvoja pitäisi pystyä muokkaamaan ilman ulkoista laitetta, ja niiden tulisi olla helposti operaattorin muokattavissa. Jos riittävän käyttöliittymän saa tehtyä ohjelmaan rajoittamatta liikaa suorituskykyä, piirikorttiohjaus olisi varteenotettava vaihtoehto kalliimpien ohjelmoitavien logiikoiden sijaan. Piirikortin dynaaminen muisti voi olla tässä kohdassa kuitenkin ratkaiseva tekijä, sillä jo pelkästään ohjelmakoodin muuttujat vievät kahdeksan prosenttia dynaamisesta muistista.

Moderneja automaatioratkaisuja ajatellessa piirikorttiohjauksella toteutetut ratkaisut jäävät auttamatta helppokäyttöisempien, varmempien ja käyttäjäläheisempien ohjelmoitavien logiikoiden varjoon, eivätkä tarjoa tarpeeksi etuja PLC-laitteisiin verrattuna.

LÄHTEET

Arduino Mega2560. Viitattu 5.1.2018. https://www.arduino.cc/en/Main/arduinoBoardMega2560/#

Ditch W. 2017. PlcLib(Arduino) User Guide

Koskinen J, 2004, Mikrotietokonetekniikka: Sulautetut järjestelmät, Otava.

Language Reference. Viitattu 5.1.2018. https://www.arduino.cc/en/Reference/HomePage

RepRapDiscount Full Graphic Smart Controller. Viitattu 5.1.2018. http://reprap.org/wiki/RepRapDiscount_Full_Graphic_Smart_Controller#Schematics

Ohjelmakoodi

```
// J. Alanne 5.1.2018

#define noPinDefs
#include <plcLib.h>

//---------------------------------CONSTANTS-------------------------------------------------------------------------------

// tulopinnit:
const int ANTURI1 = 24;
const int ANTURI2 = 26;
const int ANTURI3 = 28;
const int ANTURI4 = 30;
const int ANTURI5 = 34;
const int ANTURI6 = 36;
const int ANTURI7 = 38;
const int ANTURI8 = 40;
const int ANTURI9 = 42;
const int ANTURI10 = 44;
const int ANTURI11 = 46;
const int ANTURI12 = 48;
const int ANTURI13 = 50;
const int ANTURI14 = 52;
const int ANTURI15 = 51;
const int ANTURI16 = 49;
const int STOPPI = 41;
const int ENCODER_VASEN = 31;      //encoder CW
const int ENCODER_OIKEA = 33;      //encoderi CCW
const int ENCODER_PAINO = 35;      //encoderin nappi


// lähtöpinnit:

const int ENABLE =  12;
const int RELE1 =  2;
const int RELE2 =  22;
const int RELE3 =  4;
const int RELE4 =  3;
const int RELE5 =  6;
const int RELE6 =  5;
const int RELE7 =  8;
const int RELE8 =  7;
const int RELE9 =  10;
const int RELE10 =  9;
const int RELE11 =  13;
const int RELE12 =  11;
const int PIEZO = 37;


//-------------------------------------TIMERS----------------------------------------------------------------------------

unsigned long TIMER0135 = 0;
unsigned long TIMER0136 = 0;
unsigned long TIMER0100 = 0;
unsigned long TIMER0101 = 0;
unsigned long TIMER0048 = 0;
unsigned long TIMER0102 = 0;
unsigned long TIMER0103 = 0;
unsigned long TIMER0132 = 0;
unsigned long TIMER0104 = 0;
unsigned long TIMER0131 = 0;
```

```
unsigned long TIMER0106 = 0;
unsigned long TIMER0140 = 0;
unsigned long TIMER0150 = 0;
unsigned long TIMER0107 = 0;
unsigned long TIMER0108 = 0;
unsigned long TIMER0133 = 0;
unsigned long TIMER0134 = 0;
unsigned long TIMER0110 = 0;
unsigned long TIMER0111 = 0;
unsigned long TIMER0112 = 0;
unsigned long TIMER0114 = 0;
unsigned long TIMER0115 = 0;
unsigned long TIMER0116 = 0;
unsigned long TIMER0142 = 0;
unsigned long TIMER0113 = 0;


//--------------------------------STEPS AND TRANSMISSIONS-----------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------------
unsigned int INIT = 1;
unsigned int System_ON = 1;
unsigned int OperationMode_Homing = 0;
unsigned int OperationModeAuto = 0;
unsigned int FLAG_ReadyToStop = 0;
unsigned int FALG_ReadyToStop = 0;
unsigned int FALG_PositionSensorFree = 0;


//-------------------------------------------Magasin-----------------------------------------------------------------------

unsigned int Seq_3_HomeOK = 0;
unsigned int MAG_AUTO1 = 0;
unsigned int MAG_AUTO2 = 0;
unsigned int MAG_AUTO3 = 0;
unsigned int T0135 = 0;
unsigned int T0136 = 0;
unsigned int MAG_AUTO2_1 = 0;
unsigned int MAG_AUTO3_1 = 0;
unsigned int MAG_AUTO4 = 0;
unsigned int okToPickFromMagasin = 0;


//-------------------------------------------------BoxOpenAndMove----------------------------------------------------------

unsigned int BOAM_HOME1 = 0;
unsigned int T0100 = 0;
unsigned int BOAM_HOME2 = 0;
unsigned int T0101 = 0;
unsigned int BOAM_HOME3 = 0;
unsigned int T0048 = 0;
unsigned int BOAM_HOME4 = 0;
unsigned int BOAM_AUTO1 = 0;
unsigned int Seq_1_HomeOK = 0;
unsigned int BOAM_AUTO2 = 0;
unsigned int BOAM_AUTO3 = 0;
unsigned int T0102 = 0;
unsigned int BOAM_AUTO4 = 0;
unsigned int T0103 = 0;
unsigned int BOAM_AUTO5 = 0;
unsigned int BOAM_AUTO6 = 0;
unsigned int T0132 = 0;
unsigned int BOAM_AUTO7 = 0;
unsigned int OperationMode_ProdStopReq = 0;
unsigned int T0104 = 0;
```

```
unsigned int T0131 = 0;
unsigned int BOAM_AUTO8 = 0;
unsigned int BOAM_AUTO9 = 0;
unsigned int H20 = 0;
unsigned int BOAM_AUTO10 = 0;
unsigned int T0106 = 0;
unsigned int T0140 = 0;
unsigned int BOAM_AUTO11 = 0;
unsigned int T0150 = 0;
unsigned int BOAM_AUTO12 = 0;


//-------------------------------------------------SideBeltsAndPressing-----------------------------------------------------
unsigned int SBAP_HOME1 = 0;
unsigned int HS_GluingReadyToReceive = 0;
unsigned int HS_BoxComing = 0;
unsigned int SBAP_HOME2 = 0;
unsigned int T0107 = 0;
unsigned int Seq_2_HomeOK = 0;
unsigned int SBAP_HOME3 = 0;
unsigned int SBAP_AUTO1 = 0;
unsigned int SBAP_AUTO2 = 0;
unsigned int SBAP_AUTO3 = 0;
unsigned int SBAP_AUTO4 = 0;
unsigned int SBAP_AUTO5 = 0;
unsigned int T0108 = 0;
unsigned int SBAP_AUTO2_1_1 = 0;
unsigned int SBAP_AUTO2_2_1 = 0;
unsigned int T0133 = 0;
unsigned int SBAP_AUTO2_1_2 = 0;
unsigned int T0134 = 0;
unsigned int SBAP_AUTO2_2_2 = 0;
unsigned int SBAP_AUTO2_3 = 0;
unsigned int SBAP_AUTO2_4 = 0;
unsigned int BoxOnTriggerLiimaus1 = 0;
unsigned int T0110 = 0;
unsigned int T0111 = 0;
unsigned int T0112 = 0;
unsigned int SBAP_AUTO3_1 = 0;
unsigned int SBAP_AUTO3_2 = 0;
unsigned int SBAP_AUTO4_1 = 0;
unsigned int SBAP_AUTO4_2 = 0;
unsigned int BoxOnTriggerLiimaus2 = 0;
unsigned int T0114 = 0;
unsigned int T0115 = 0;
unsigned int T0116 = 0;
unsigned int SBAP_AUTO5_1 = 0;
unsigned int SBAP_AUTO5_2 = 0;
unsigned int SBAP_AUTO6 = 0;
unsigned int T1042 = 0;
unsigned int SBAP_AUTO7 = 0;
unsigned int T0113 = 0;
unsigned int HS_RelayK009 = 0;
unsigned int SBAP_AUTO8 = 0;
unsigned int SBAP_AUTO9 = 0;
unsigned int SBAP_AUTO10 = 0;



//----------------------------SETUP-------------------------------------------------------------------------------------------
//----------------------------------------------------------------------------------------------------------------------------

void setup() {
```

```
  void customIO();
}

void customIO() {

  //lähtöpinnit outputeiksi
  pinMode(RELE1, OUTPUT);
  pinMode(RELE2, OUTPUT);
  pinMode(RELE3, OUTPUT);
  pinMode(RELE4, OUTPUT);
  pinMode(RELE5, OUTPUT);
  pinMode(RELE6, OUTPUT);
  pinMode(RELE7, OUTPUT);
  pinMode(RELE8, OUTPUT);
  pinMode(RELE9, OUTPUT);
  pinMode(RELE10, OUTPUT);
  pinMode(RELE11, OUTPUT);
  pinMode(ENABLE, OUTPUT);
  pinMode(PIEZO, OUTPUT);

  // tulopinnit inputeiksi:
  pinMode(ANTURI1, INPUT);
  pinMode(ANTURI2, INPUT);
  pinMode(ANTURI3, INPUT);
  pinMode(ANTURI4, INPUT);
  pinMode(ANTURI5, INPUT);
  pinMode(ANTURI6, INPUT);
  pinMode(ANTURI7, INPUT);
  pinMode(ANTURI8, INPUT);
  pinMode(ANTURI9, INPUT);
  pinMode(ANTURI10, INPUT);
  pinMode(ANTURI11, INPUT);
  pinMode(ANTURI12, INPUT);
  pinMode(ANTURI13, INPUT);
  pinMode(ANTURI14, INPUT);
  pinMode(ANTURI15, INPUT);
  pinMode(ANTURI16, INPUT);
  pinMode(ENCODER_VASEN, INPUT);
  pinMode(ENCODER_OIKEA, INPUT);
  pinMode(ENCODER_PAINO, INPUT);
  pinMode(STOPPI, INPUT);
}

//--------------OHJELMAKOODIN  ALKU---------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------------

void loop() {
  loopSideBeltsAndPressing();
  loopBoxOpenAndMove();
  loopMagasin();

}




//-----------------------------------------SIDE BELTS AND PRESSING koodi alkaa tästä-------------------------------
//------------------------------------------------------------------------------------------------------------------------


void loopSideBeltsAndPressing(){
```

```
in(INIT);
set(OperationMode_Homing);
set(SBAP_HOME1);

in(SBAP_HOME1);
reset(RELE1); //GLUE GUN (XY1_1)
reset(RELE6); //BOTTOM PRESSER (XY1_Y5)
reset(RELE7); //UPPER PRESSER (XY1_Y6)

in(SBAP_HOME1);
andBit(OperationMode_Homing);
andBit(ANTURI12); //GLUE UPPER PRESSER DOWN
andBit(ANTURI10); //GLUE BOTTOM PRESSER DOWN
set(SBAP_HOME2);
reset(SBAP_HOME1);

in(SBAP_HOME2);
out(RELE12); //OUTFEED BELT U15 2 FWD
out(RELE11); //OUTFEED BELT U14 1 FWD

in(SBAP_HOME2);
andBit(OperationMode_Homing);
andNotBit(ANTURI9); //GLUE TRIGGER (XB1_B9)
andNotBit(ANTURI13); //BOX AT PRESSING POS (XB1_B13)
timerOn(TIMER0107, 2000);
out(T0107);

in(SBAP_HOME2);
andBit(OperationMode_Homing);
andBit(T0107);
set(SBAP_HOME3);
reset(SBAP_HOME2);

in(SBAP_HOME3);
andBit(OperationMode_Homing);
set(SBAP_AUTO1);
reset(SBAP_HOME3);

in(SBAP_AUTO1);
andBit(System_ON);
set(Seq_2_HomeOK);
set(HS_GluingReadyToReceive);
set(FLAG_ReadyToStop);

in(SBAP_AUTO1);
andBit(OperationModeAuto);
andBit(HS_BoxComing);
set(SBAP_AUTO2);
set(SBAP_AUTO3);
set(SBAP_AUTO4);
set(SBAP_AUTO5);
reset(SBAP_AUTO1);

//-------------AUTO2 HAARA-----------------------------------------------------------------------------------------------------------

in(SBAP_AUTO2);
andBit(OperationModeAuto);
out(RELE12); //OUTFEED BELT U15 2 FWD (U15_FWD)
out(RELE11); //OUTFEED BELT U14 1 FWD
timerOn(TIMER0108, 2000);
out(T0108);
reset(HS_GluingReadyToReceive);
reset(FLAG_ReadyToStop);
```

```
in(SBAP_AUTO2);
andBit(T0108);
orBit(ANTURI13); //BOX AT PRESSING POS (XB1_B13)
andBit(FALG_PositionSensorFree);
set(SBAP_AUTO2_1_1);
set(SBAP_AUTO2_2_1);
reset(SBAP_AUTO2);

in(SBAP_AUTO2_1_1);
andBit(OperationModeAuto);
out(RELE11); //OUTFEED BELT U14 1 FWD

in(SBAP_AUTO2_1_1);
andBit(OperationModeAuto);
andBit(FALG_PositionSensorFree);
timerOn(TIMER0133, 2000);
out(T0133);

in(SBAP_AUTO2_1_1);
andBit(ANTURI13); //BOX AT PRESSING POS (XB1_B13)
andBit(T0133);
set(SBAP_AUTO2_1_2);
reset(SBAP_AUTO2_1_1);

in(SBAP_AUTO2_2_1);
andBit(OperationModeAuto);
out(RELE12); //OUTFEED BELT U15 2 FWD

in(SBAP_AUTO2_2_1);
andBit(OperationModeAuto);
andBit(ANTURI13); //BOX AT PRESSING POS (XB1_B13)
andBit(FALG_PositionSensorFree);
timerOn(TIMER0134, 2000);
out(T0134);

in(SBAP_AUTO2_2_1);
andBit(T0134);
set(SBAP_AUTO2_2_2);
reset(SBAP_AUTO2_2_1);

in(SBAP_AUTO2_1_2);
andBit(SBAP_AUTO2_2_2);
set(SBAP_AUTO2_3);
reset(SBAP_AUTO2_1_2);
reset(SBAP_AUTO2_2_2);

in(SBAP_AUTO2_3);
andBit(OperationModeAuto);
set(SBAP_AUTO2_4);
reset(SBAP_AUTO2_3);

//----------------AUTO3 haara-----------------------------------------------------------------------------------------------------------

in(SBAP_AUTO3);
andBit(OperationModeAuto);
andBit(ANTURI9); //GLUE TRIGGER (XB1_B9)
set(BoxOnTriggerLiimaus1);

in(SBAP_AUTO3);
andBit(OperationModeAuto);
andBit(BoxOnTriggerLiimaus1);
timerOn(TIMER0110, 2000);
out(T0110);
```

```
in(SBAP_AUTO3);
andBit(T0110);
set(SBAP_AUTO3_1);
reset(SBAP_AUTO3);

in(SBAP_AUTO3_1);
andBit(System_ON);
timerOn(TIMER0111, 2000);
timerOn(TIMER0112, 2000);
set(T0111);
set(T0112);

in(SBAP_AUTO3_1);
andBit(System_ON);
andNotBit(T0112);
andBit(OperationModeAuto);
andBit(SBAP_AUTO2_1_1);
orBit(SBAP_AUTO2);
out(RELE1); //GLUE GUN (XY1_1)

in(SBAP_AUTO3_1);
andBit(OperationModeAuto);
andBit(T0111);
set(SBAP_AUTO3_2);
reset(SBAP_AUTO3_1);

in(SBAP_AUTO3_2);
andBit(OperationModeAuto);
reset(BoxOnTriggerLiimaus1);

//--------------AUTO4 haara-----------------------------------------------------------------------------------------------------

in(SBAP_AUTO4);
andBit(OperationModeAuto);
andNotBit(ANTURI13); //BOX AT PRESSING POS. (XB1_B13)
set(SBAP_AUTO4_1);
reset(SBAP_AUTO4);

in(SBAP_AUTO4_1);
andBit(System_ON);
set(FALG_PositionSensorFree);

//---------------AUTO5 haara-----------------------------------------------------------------------------------------------------

in(SBAP_AUTO5);
andBit(OperationModeAuto);
andBit(ANTURI9); //GLUE TRIGGER (XB1_B9)
set(BoxOnTriggerLiimaus2);

in(SBAP_AUTO5);
andBit(OperationModeAuto);
andBit(BoxOnTriggerLiimaus2);
timerOn(TIMER0114, 2000);
out(T0114);

in(SBAP_AUTO5);
andBit(System_ON);
andBit(T0114);
set(SBAP_AUTO5_1);
reset(SBAP_AUTO5);

in(SBAP_AUTO5_1);
```

```
andBit(System_ON);
timerOn(TIMER0115, 2000);
timerOn(TIMER0116, 2000);
set(T0115);
set(T0116);

in(SBAP_AUTO5_1);
andBit(System_ON);
andNotBit(T0116);
andBit(OperationModeAuto);
andBit(SBAP_AUTO2_1_1);
orBit(SBAP_AUTO2);
out(RELE1); //GLUE GUN (XY1_1)

in(SBAP_AUTO5_1);
andBit(OperationModeAuto);
andBit(T0115);
set(SBAP_AUTO5_2);
reset(SBAP_AUTO5_1);

in(SBAP_AUTO5_2);
reset(BoxOnTriggerLiimaus2);

//----------TRANSMISSION to AUTO6----------------------------------------------------------------------------------------

in(SBAP_AUTO2_4);
andBit(SBAP_AUTO3_2);
andBit(SBAP_AUTO4_1);
andBit(SBAP_AUTO5_2);
set(SBAP_AUTO6);
reset(SBAP_AUTO2_4);
reset(SBAP_AUTO3_2);
reset(SBAP_AUTO4_1);
reset(SBAP_AUTO5_2);

in(SBAP_AUTO6);
andBit(OperationModeAuto);
set(RELE7); //UPPER PRESSER (XY1_Y6)
set(RELE6); //BOTTOM PRESSER (XY1_Y5)
reset(FALG_PositionSensorFree);

in(SBAP_AUTO6);
andNotBit(FALG_PositionSensorFree);
timerOn(TIMER0142, 2000);
set(T1042);

in(SBAP_AUTO6);
andBit(ANTURI11);
orBit(T1042);
set(SBAP_AUTO7);
reset(SBAP_AUTO6);

in(SBAP_AUTO7);
andBit(OperationModeAuto);
timerOn(TIMER0113, 2000);
set(T0113);

in(SBAP_AUTO7);
andBit(OperationModeAuto);
andBit(T0113);
set(SBAP_AUTO8);
reset(SBAP_AUTO7);

in(SBAP_AUTO8);
```

```
  andBit(OperationModeAuto);
  reset(RELE7); //UPPER PRESSER (XY1_Y6)
  reset(RELE6); //BOTTOM PRESSER (XY1_Y5)

  in(SBAP_AUTO8);
  andBit(OperationModeAuto);
  andBit(ANTURI12); //GLUE UPPER PRESSER DOWN (XB1_B12)
  andBit(ANTURI10); //GLUE BOTTOM PRESSER DOWN (XB1_B10)
  set(SBAP_AUTO9);
  reset(SBAP_AUTO8);

  in(SBAP_AUTO9);
  andBit(OperationModeAuto);
  reset(HS_BoxComing);
  out(FLAG_ReadyToStop);

  in(SBAP_AUTO9);
  andBit(OperationModeAuto);
  andBit(ANTURI14);
  andNotBit(HS_RelayK009);
  andNotBit(OperationMode_ProdStopReq);
  set(SBAP_AUTO10);
  reset(SBAP_AUTO9);

  in(SBAP_AUTO10);
  andBit(OperationModeAuto);
  andNotBit(ANTURI9); //GLUE TRIGGER (XB1_B9)
  set(SBAP_AUTO1);
  reset(SBAP_AUTO10);
}


//---------------------------------------------BOX OPEN AND MOVE koodi alkaa tästä-----------------------------------
//-------------------------------------------------------------------------------------------------------------------------------------

void loopBoxOpenAndMove(){

  in(INIT);
  set(OperationMode_Homing);
  set(BOAM_HOME1);

  in(BOAM_HOME1);
  reset(RELE2); //BLANK SUCTION (XY1_Y1)
  andBit(OperationMode_Homing);
  timerOn(TIMER0100, 2000);
  out(T0100);

  in(BOAM_HOME1);
  andBit(OperationMode_Homing);
  andBit(T0100);
  set(BOAM_HOME2);
  reset(BOAM_HOME1);

  in(BOAM_HOME2);
  reset(RELE4); //BLANK PICK (XY1_Y3)
  set(RELE3); //BLANK OPENING NAIL (XY1_Y2)
  timerOn(TIMER0101, 2000);
  out(T0101);

  in(BOAM_HOME2);
  andBit(OperationMode_Homing);
  andBit(ANTURI6);
  andBit(T0101);
```

```
set(BOAM_HOME3);
reset(BOAM_HOME2);

in(BOAM_HOME3);
andBit(OperationMode_Homing);
reset(RELE5); //BLANK FLAP TILT (XY1_Y4)
out(RELE10); //BOX MOVER U13_FWD

in(BOAM_HOME3);
andNotBit(ANTURI8); //BOX MOVE AT BACK (XB1_B8)
timerOn(TIMER0048, 5000);
out(T0048);


in(BOAM_HOME3);
andBit(OperationMode_Homing);
andBit(T0048);
set(BOAM_HOME4);
reset(BOAM_HOME3);

in(BOAM_HOME4);
andBit(OperationMode_Homing);
out(RELE10); //BOX MOVER U13_BWD

in(BOAM_HOME4);
andBit(OperationMode_Homing);
andBit(ANTURI8); //BOX MOVE AT BACK (XB1_B8)
set(BOAM_AUTO1);
reset(BOAM_HOME4);

in(BOAM_AUTO1);
andBit(System_ON);
set(Seq_1_HomeOK);
out(FALG_ReadyToStop);

in(BOAM_AUTO1);
andBit(okToPickFromMagasin);
andBit(OperationModeAuto);
andBit(ANTURI14); //RUSH AT OUTFEED (XB1_B14)
set(BOAM_AUTO2);
reset(BOAM_AUTO1);

in(BOAM_AUTO2);
andBit(OperationModeAuto);
set(RELE2); //BLANK SUCTION (XY1_Y1)
set(RELE4); //BLANK PICK (XY1_Y3)
reset(RELE3); //BLANK OPENING NAIL (XY1_Y2)

in(BOAM_AUTO2);
andBit(OperationModeAuto);
andBit(RELE5); //BLANK PICK AT FRONT (XB1_B5)
set(BOAM_AUTO3);
reset(BOAM_AUTO2);

in(BOAM_AUTO3);
andBit(OperationModeAuto);
timerOn(TIMER0102, 2000);
out(T0102);

in(BOAM_AUTO3);
andBit(OperationModeAuto);
andBit(T0102);
set(BOAM_AUTO4);
reset(BOAM_AUTO3);
```

```
in(BOAM_AUTO4);
reset(RELE4); //BLANK PICK (XY1_Y3)
timerOn(TIMER0103, 2000);
out(T0103);

in(BOAM_AUTO4);
andBit(T0103);
set(RELE3); //BLANK OPENING NAIL (XY1_Y2)

in(BOAM_AUTO4);
andBit(OperationModeAuto);
andBit(ANTURI6); // BLANK PICK AT BACK (XB1_B6)
andBit(RELE4); //BLANK PICK (XY1_Y3)
set(BOAM_AUTO5);
reset(BOAM_AUTO4);

in(BOAM_AUTO5);
andBit(OperationModeAuto);
reset(RELE3); //BLANK OPENING NAIL (XY1_Y2)
set(BOAM_AUTO6);
reset(BOAM_AUTO5);

in(BOAM_AUTO6);
andBit(OperationModeAuto);
set(RELE5); //BACK FLAP TILT (XY1_Y2)
out(FALG_ReadyToStop);
timerOn(TIMER0132, 2000);
out(T0132);

in(BOAM_AUTO6);
andBit(HS_GluingReadyToReceive); //SideBeltsAndPressing koodissa
andBit(OperationModeAuto);
andBit(T0132);
andNotBit(HS_BoxComing); //SideBeltsAndPressing koodissa
andNotBit(OperationMode_ProdStopReq);
andBit(ANTURI14); //RUSH AT OUTFEED (XB1_B14)
set(BOAM_AUTO7);
reset(BOAM_AUTO6);

in(BOAM_AUTO7);
set(HS_BoxComing);
andBit(OperationModeAuto);
out(RELE10); //BOX MOVER U13 FWD
timerOn(TIMER0104, 2000);
timerOn(TIMER0131, 2000);
out(T0104);
out(T0131);

in(BOAM_AUTO7);
andBit(T0131);
reset(RELE2); //BLANK SUCTION

in(BOAM_AUTO7);
andBit(T0104);
set(BOAM_AUTO8);
reset(BOAM_AUTO7);

in(BOAM_AUTO8);
andBit(OperationModeAuto);
out(RELE10); // BOX MOVER U13 FWD
reset(RELE2); //BLANK SUCTION

in(BOAM_AUTO8);
```

```
andBit(OperationModeAuto);
andBit(ANTURI7); //BOX MOVE AT FRONT
set(BOAM_AUTO9);
reset(BOAM_AUTO8);

in(BOAM_AUTO9);
reset(H20);

in(BOAM_AUTO9);
andBit(OperationModeAuto);
set(BOAM_AUTO10);
reset(BOAM_AUTO9);

in(BOAM_AUTO10);
andBit(System_ON);
andNotBit(OperationModeAuto);
set(H20);

in(BOAM_AUTO10);
andBit(System_ON);
andBit(OperationModeAuto);
out(RELE10); // BOX MOVER U13 BWD
timerOn(TIMER0106, 2000);
timerOn(TIMER0140, 2000);
out(T0106);
out(T0140);

in(BOAM_AUTO10);
andBit(T0140);
reset(RELE5); //BACK FLAP TILT

in(BOAM_AUTO10);
andBit(T0106);
andNotBit(RELE5);
set(BOAM_AUTO11);
reset(BOAM_AUTO10);

in(BOAM_AUTO11);
andBit(System_ON);
andNotBit(OperationModeAuto);
set(H20);

in(BOAM_AUTO11);
andBit(System_ON);
andBit(OperationModeAuto);
out(RELE10); //BOX MOVER U13 BWD
timerOn(TIMER0150, 2000);
out(T0150);

in(BOAM_AUTO11);
andBit(OperationModeAuto);
andBit(T0150);
andNotBit(H20);
set(RELE4); //BLANK PICK

in(BOAM_AUTO11);
andBit(OperationModeAuto);
andBit(ANTURI8); // BOX MOVE AT BACK
set(BOAM_AUTO12);
reset(BOAM_AUTO11);

in(BOAM_AUTO12);
andBit(OperationModeAuto);
andNotBit(HS_BoxComing);
```

```
  set(BOAM_AUTO1);
  reset(BOAM_AUTO12);

  in(BOAM_AUTO12);
  andBit(OperationModeAuto);
  andBit(HS_BoxComing);
  set(BOAM_AUTO1);
  reset(BOAM_AUTO12);

}

//-----------------------------------------------------MAGASIN koodi alkaa tästä-----------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------------------

void loopMagasin(){

  in(INIT);
  set(Seq_3_HomeOK);
  set(OperationModeAuto);
  set(MAG_AUTO1);
  reset(INIT);

  in(MAG_AUTO1);
  andBit(OperationModeAuto);
  andBit (System_ON);
  set(MAG_AUTO2);
  set(MAG_AUTO3);
  reset(MAG_AUTO1);

  in(MAG_AUTO2);
  andBit(ANTURI4);  //BLANKS EMPTY (XB1_B4)
  out(RELE9);  //MAGASIN RIGHT BELT FWD (U_12)

  in(MAG_AUTO2);
  andBit(ANTURI1);  //RIGHT SIDE MAGASIN OK (XB1_B1)
  timerOn(TIMER0135, 2000);
  out(T0135);

  in(MAG_AUTO2);
  andBit(ANTURI1); //RIGHT SIDE MAGASIN OK (XB1_B1)
  andBit(T0135);
  set(MAG_AUTO2_1);
  reset(MAG_AUTO2);

  in(MAG_AUTO3);
  andBit(ANTURI4);  //BLANKS EMPTY (XB1_B4)
  out(RELE8);  //MAGASIN LEFT BELT FWD (U_11)

  in(MAG_AUTO3);
  andBit(ANTURI2);  //LEFT SIDE MAGASIN OK (XB1_B12)
  timerOn(TIMER0136, 2000);
  out(T0136);

  in(MAG_AUTO3);
  andBit(ANTURI2); //LEFT SIDE MAGASIN OK (XB1_B12)
  andBit(T0136);
  set(MAG_AUTO3_1);
  reset(MAG_AUTO3);

  in(MAG_AUTO2_1);
  andBit(MAG_AUTO3_1);
  andBit(OperationModeAuto);
  set(MAG_AUTO4);
  reset(MAG_AUTO2_1);
```

```
  reset(MAG_AUTO3_1);

  in(MAG_AUTO4);
  andBit(ANTURI4);
  andBit(ANTURI1);
  andBit(ANTURI2);
  out(okToPickFromMagasin);

  in(MAG_AUTO4);
  andNotBit(ANTURI1);
  orNotBit(ANTURI2);
  set(MAG_AUTO1);
  reset(MAG_AUTO4);

}
```